

Chapter 2: 8051 Software Assembly Language Programming

(中文教材第3章)

The 8051 Microcontroller and Embedded Systems

RE & EE, BJTU

Fen Tang (唐芬), Hong Jeong

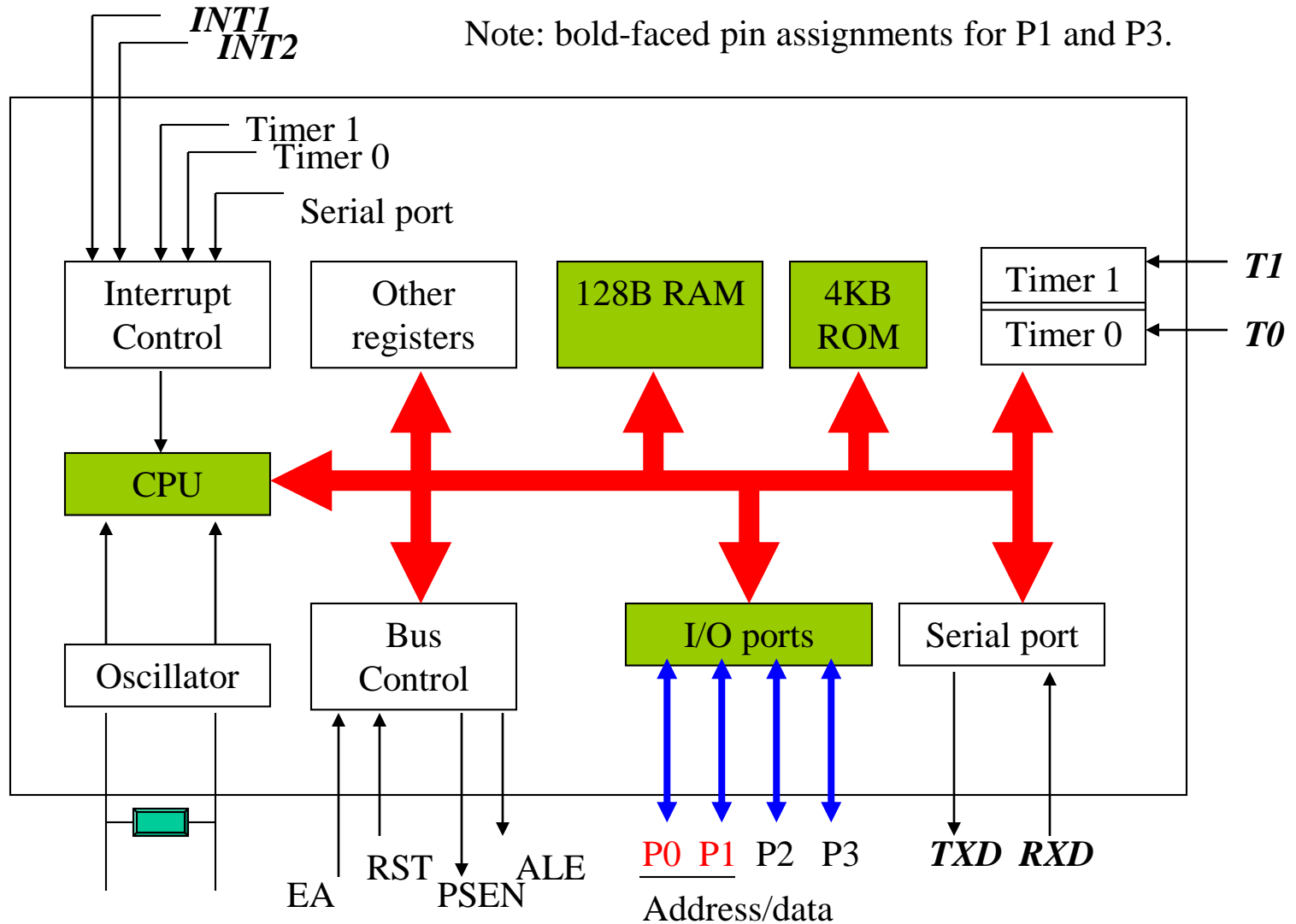
Course overview

- Chapter 1: 8051 hardware/architecture
- Chapter 2: 8051 software/ Assembly Language Programming
- Chapter 3: Jump, Loop and Call Instructions
- Chapter 4: IO Port Programming
- Chapter 5: 8031/51 Interfacing to External Memory
- Chapter 6: Interrupts programming
- Chapter 7: Timer programming
- Chapter 8: 8031/51 Interfacing with the 8155
- Chapter 9: 8031/51 Interfacing with the ADC
- Chapter 10: 8031/51 Interfacing with the DAC
- Chapter 11: Serial Communication
- Experiments

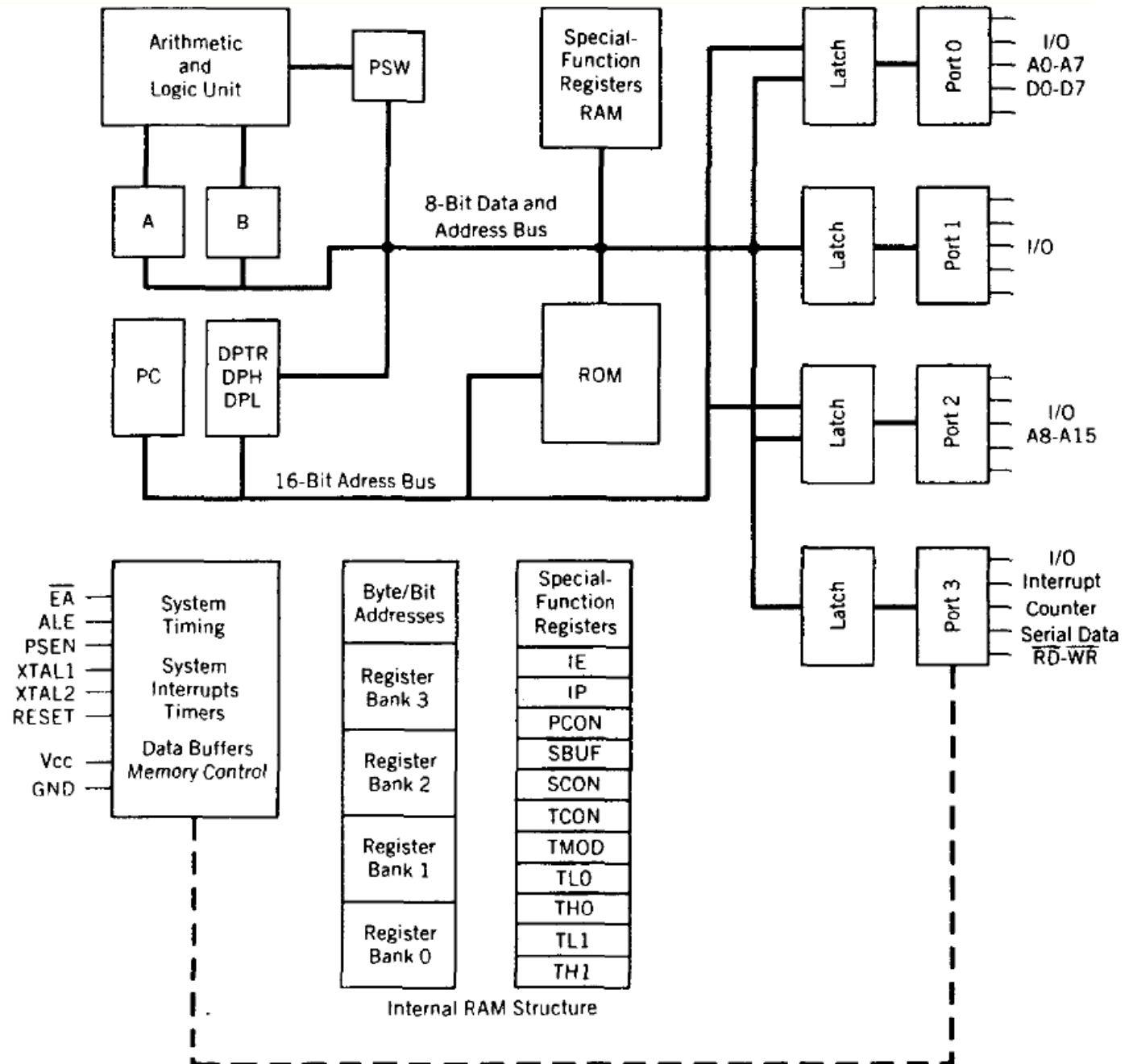
Review

- 8051 Architecture

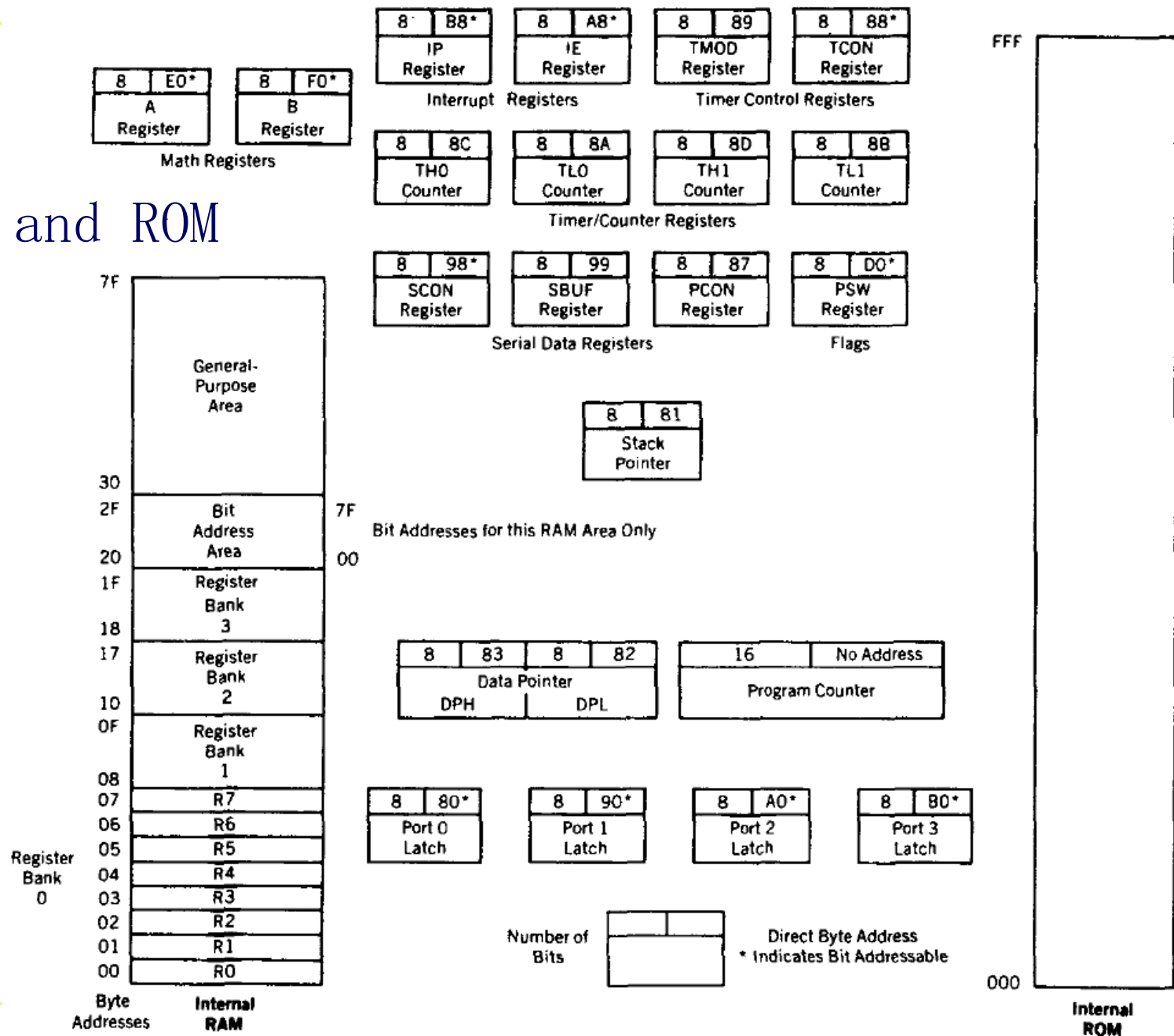
8051 Block Diagram



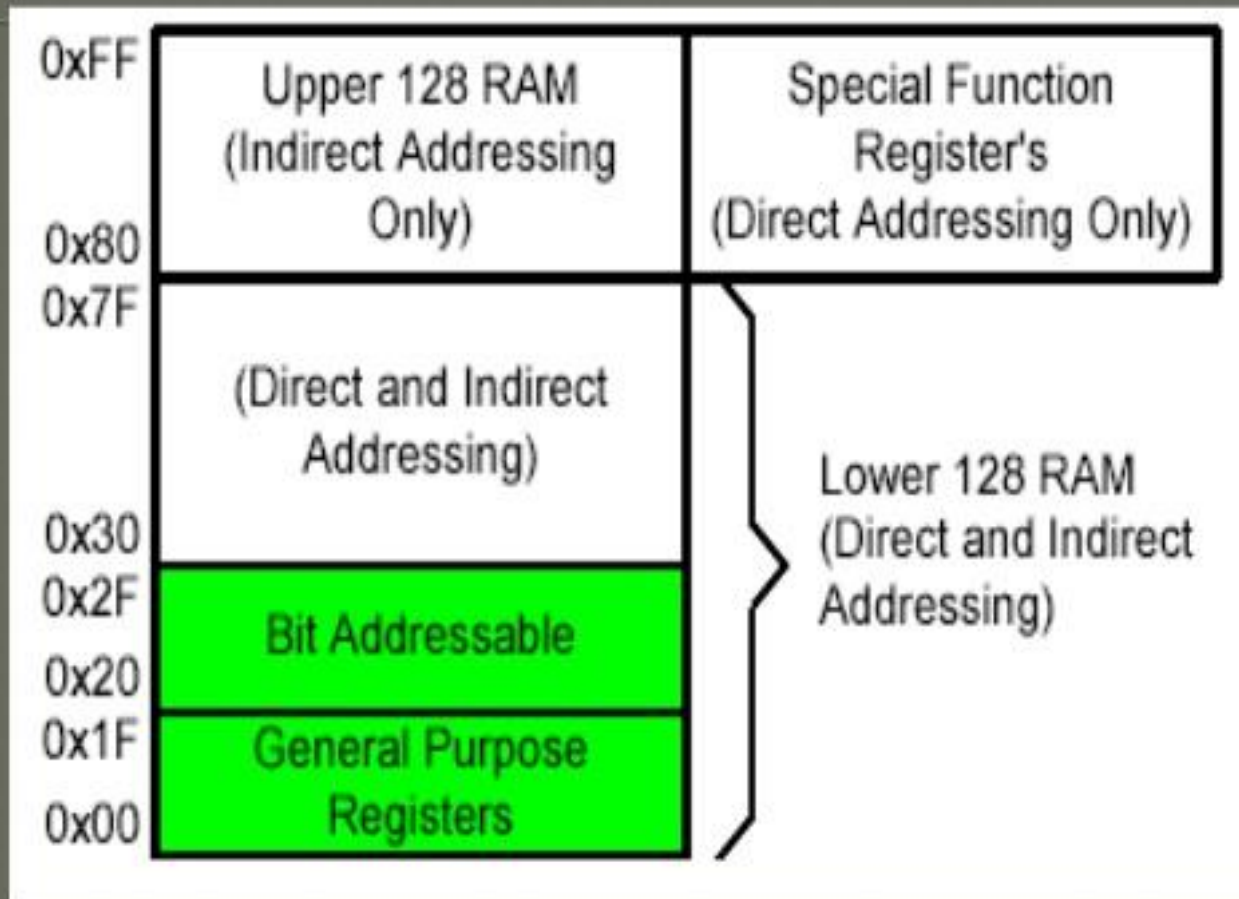
8051 Block Diagram



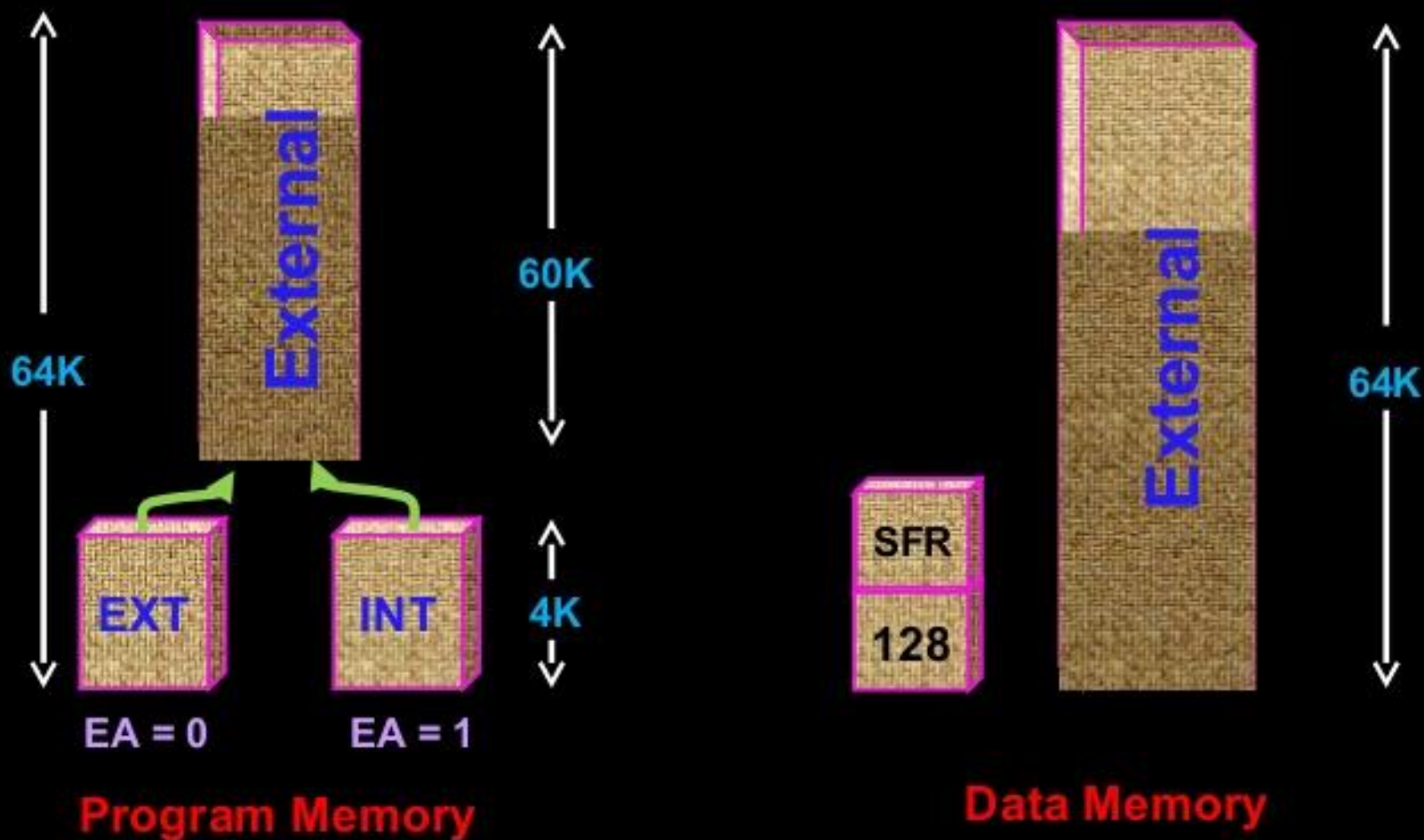
8051 RAM and ROM



On-Chip Memory-Internal RAM



8051 Memory Structure

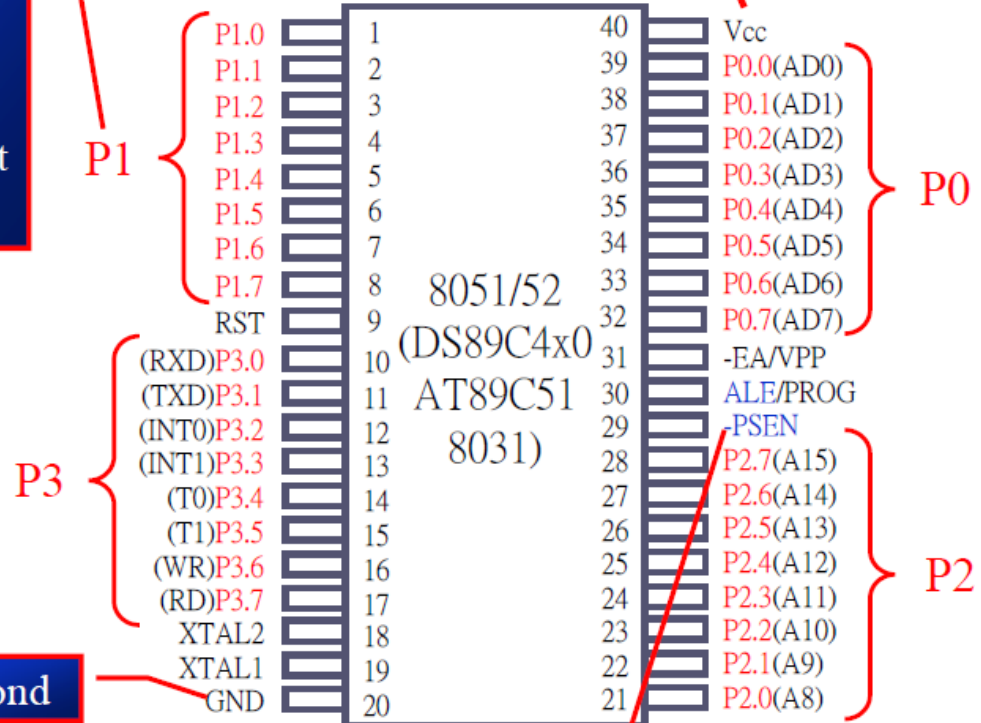


8051 Pins

A total of 32 pins are set aside for the four ports P0, P1, P2, P3, where each port takes 8 pins

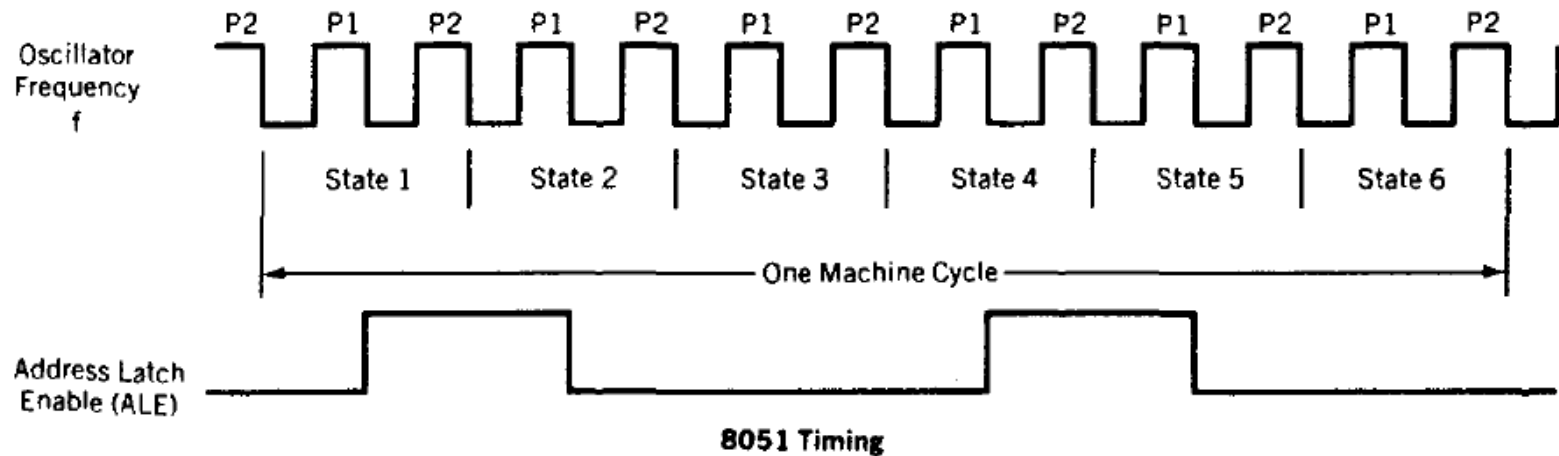
Vcc, GND, XTAL1, XTAL2, RST, -EA are used by all members of 8051 and 8031 families

Provides +5V supply voltage to the chip



-PSEN and ALE are used mainly in 8031-based systems

8051 Timing



- Q: if the crystal frequency is 6 MHz, then the frequency of ALE is 1 MHz, and the time to execute an **ADD A, R1** instruction is 2 microseconds.

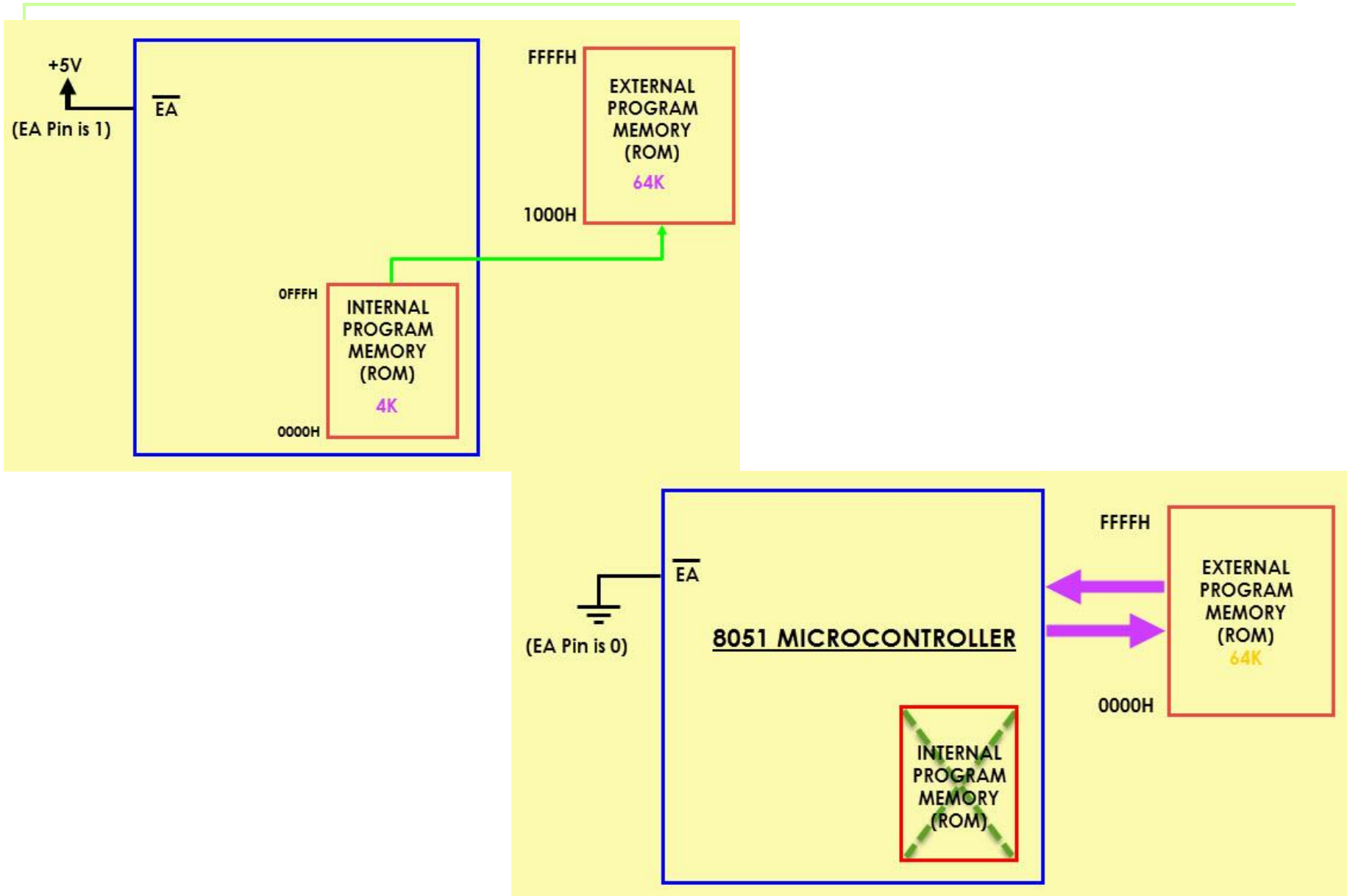
Reset Pin

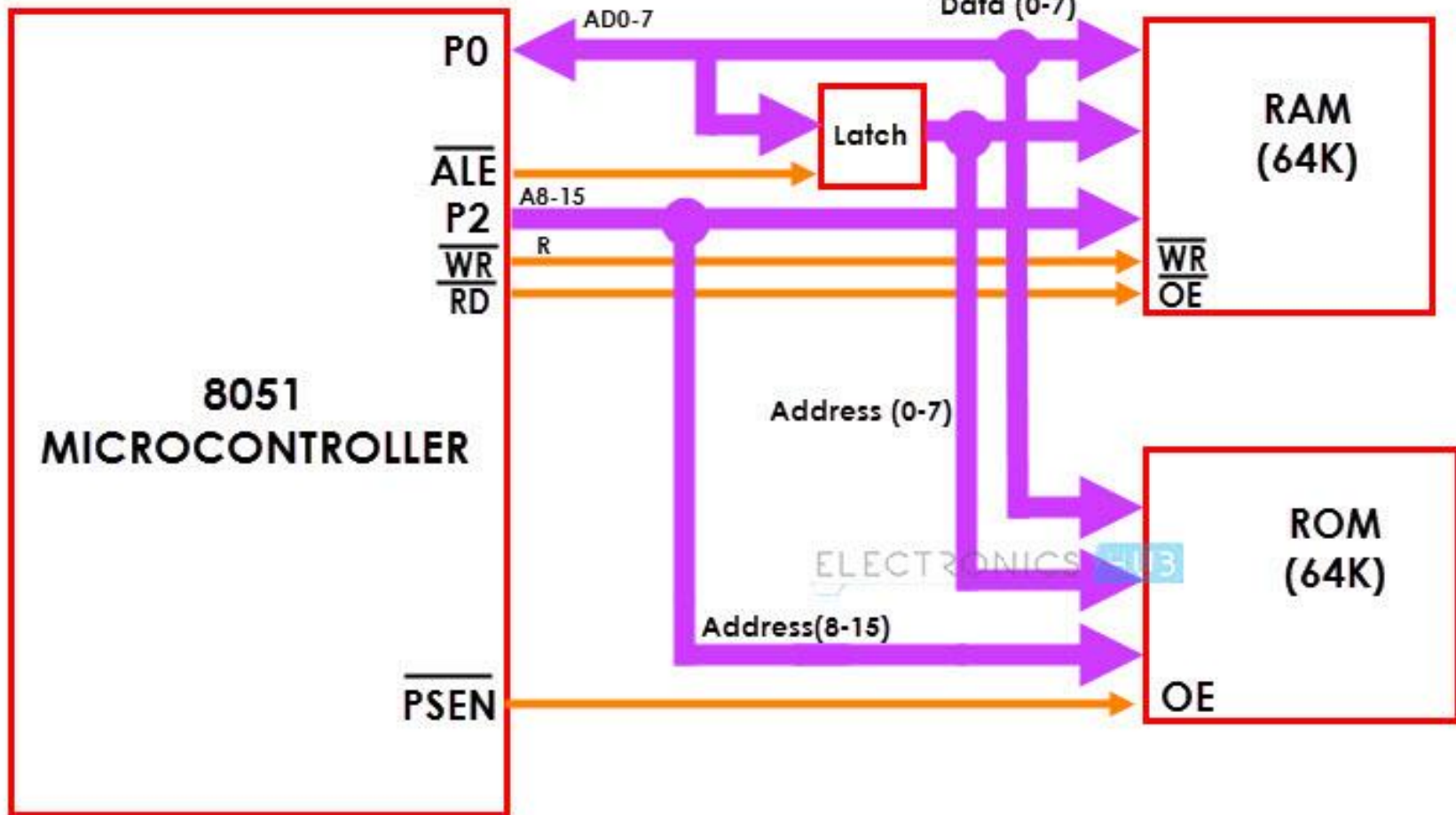
- **RESET** pin is an input and is **active high** (normally low)
 - ❑ Upon applying a **high pulse** to this pin, the microcontroller will reset and terminate all activities
 - This is often referred to as a **power-on reset**
 - Activating a power-on reset will cause all values in the registers to be lost

RESET value of some
8051 registers

we must place
the first line of
source code in
ROM location 0

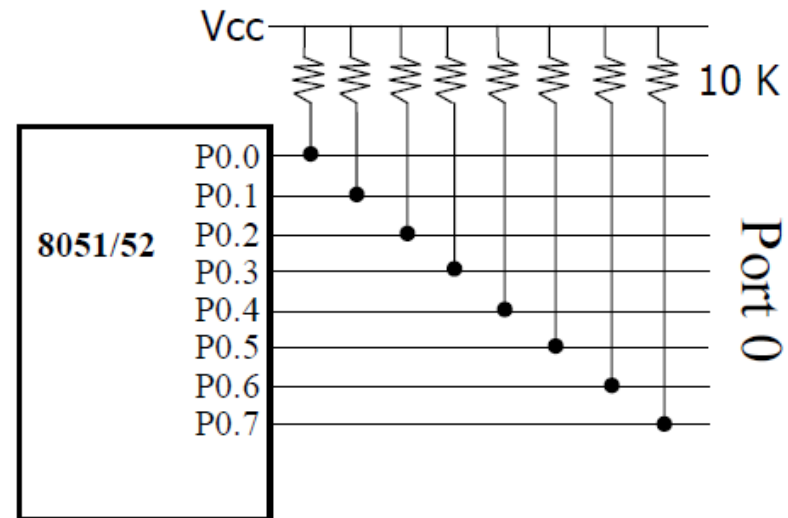
Register	Reset Value
PC	0000
DPTR	0000
ACC	00
PSW	00
SP	07
B	00
P0-P3	FF





Port0

- It can be used for input or output, each pin must be connected externally to a 10K ohm pull-up resistor
 - ❑ This is due to the fact that P0 is an open drain, unlike P1, P2, and P3
 - ❑ Open drain is a term used for MOS chips in the same way that open collector is used for TTL chips



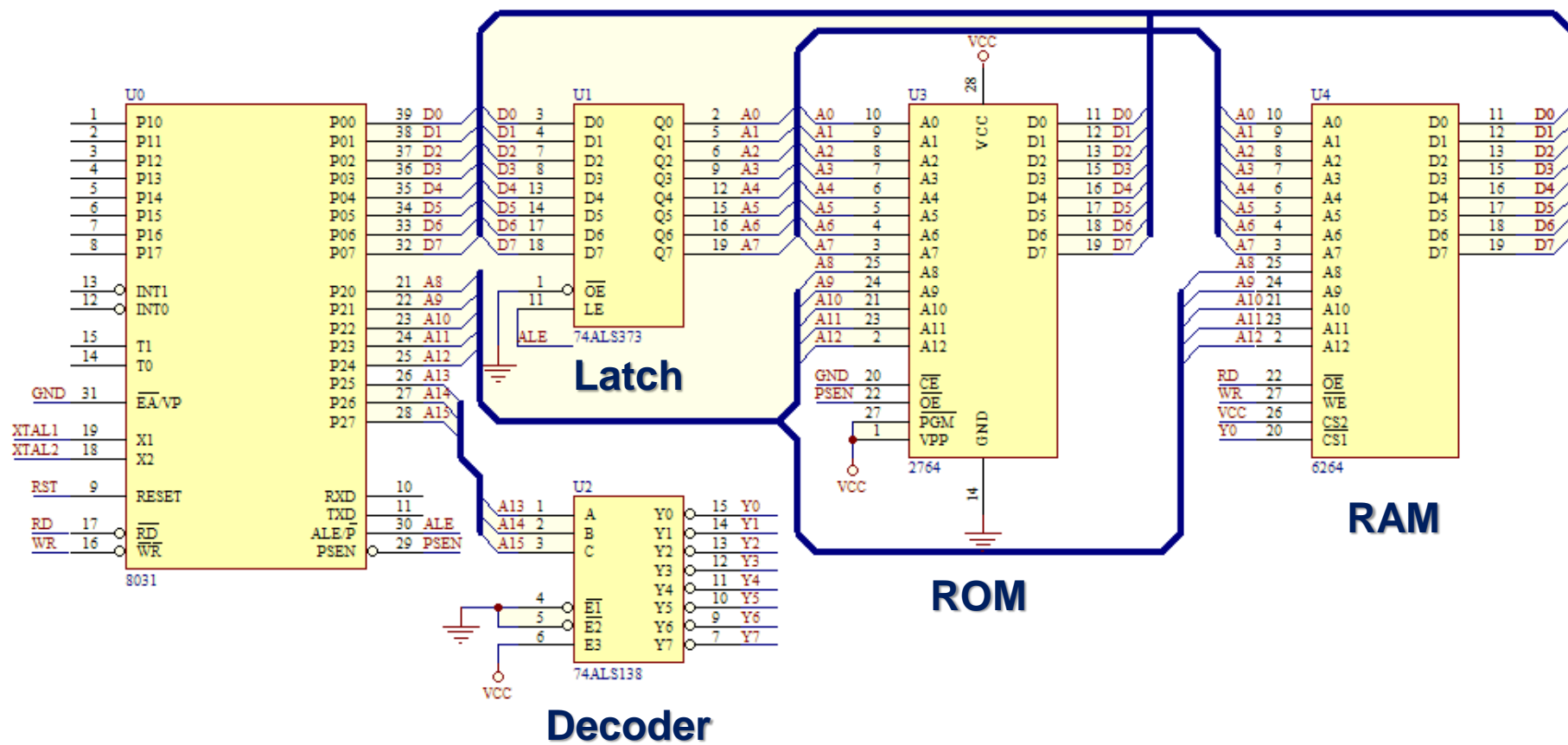
Port 1 and Port 2

- In 8051-based systems with no external memory connection
 - Both P1 and P2 are used as simple I/O
- In 8031/51-based systems with external memory connections
 - Port 2 must be used along with P0 to provide the 16-bit address for the external memory
 - P0 provides the lower 8 bits via A0 – A7
 - P2 is used for the upper 8 bits of the 16-bit address, designated as A8 – A15, and it cannot be used for I/O

Port 3

- Port 3 can be used as input or output
 - Port 3 does not need any pull-up resistors
- Port 3 has the additional function of providing some extremely important signals

P3 Bit	Function	Pin	
P3.0	RxD	10	Serial communications
P3.1	TxD	11	
P3.2	<u>INT0</u>	12	External interrupts
P3.3	<u>INT1</u>	13	
P3.4	T0	14	Timers
P3.5	T1	15	
P3.6	<u>WR</u>	16	Read/Write signals of external memories
P3.7	<u>RD</u>	17	



OUTLINES

- ➡ Inside the 8051
- ➡ Introduction to 8051 assembly programming
- ➡ Assembling and running an 8051 program
- ➡ Program counter and ROM space
- ➡ 8051 data type and directives
- ➡ Flag bits and PSW register
- ➡ 8051 register banks and stack



Inside the 8051

Register are used to **store information temporarily**, while the information could be

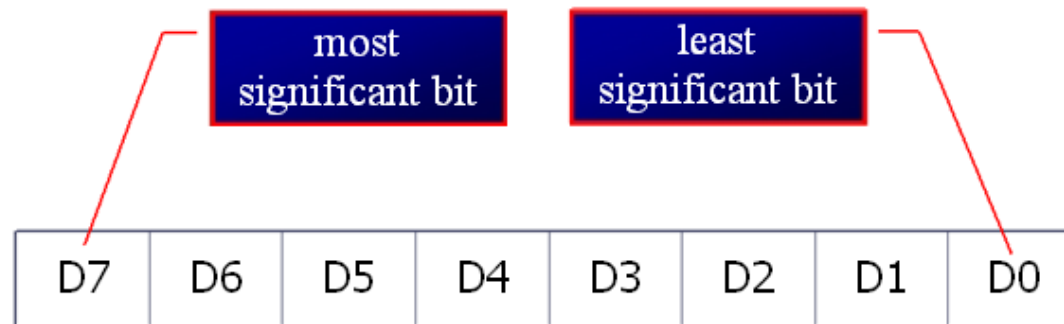
- A byte of **data** to be processed
- An **address** pointing to the data to be fetched

The vast majority of 8051 register are **8-bit register**

- Only one data type, **8 bits**

The 8 bits of a register are shown from **MSB D7** to the **LSB D0**

- With an 8-bit data type, any data **larger than 8 bits** must be **broken into** 8-bit chunks before it is processed.



8 bit Registers

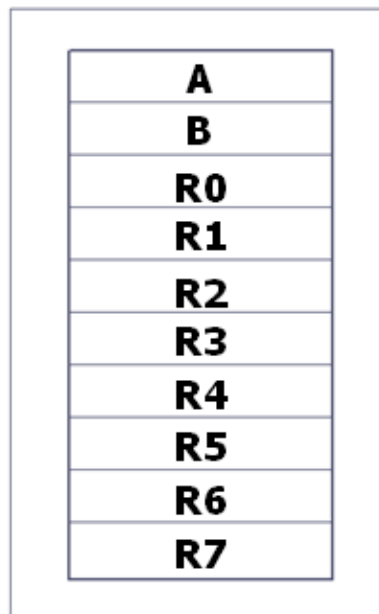
Inside the 8051

The most widely used Register

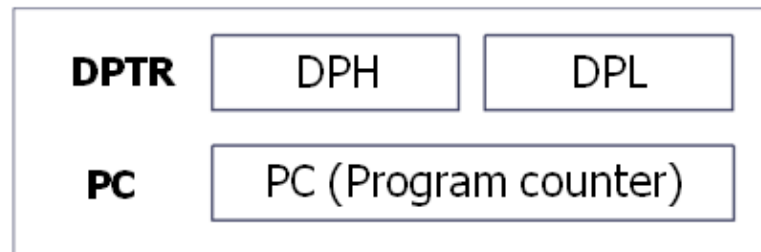
- **A (Accumulator)**: used for all **arithmetic and logic** instructions
- B, R0, R1, R2, R3, R4, R5, R6, R7
- DPTR (data pointer), and PC (program counter)

80	PSW	SP	DPL	DPH			PCON	87
88	TCON	TMOD	TL0	TL1	TH0	TH1		8F
90	P1							97
98	SCON	SBUF						9F
A0	P2							A7
A8	IE							AF
B0	P3							B7
B8	IP							B7
C0								C7
C8								CF
D0	PSW							D7
D8								DF
E0								E7
E8								EF
F0								F7
F8								FF

Blue background are I/O port SFRs
Yellow background are control SFRs
Green background are other SFRs



8-bit register



16-bit register

Inside the 8051

The 8051 contains two **16-bit** registers : **PC** and **DPTR**

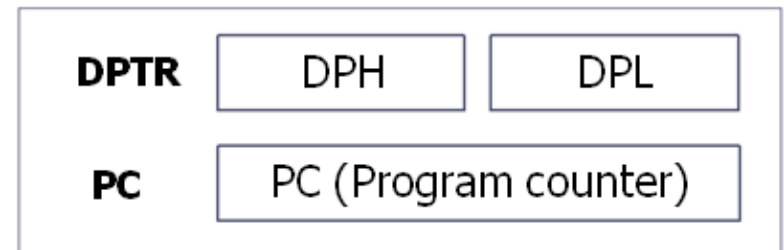
➤ **PC** (program counter)

- An instruction will be fetched from **memory location** addressed by the PC, and PC **is automatically incremented**.
- The PC is the **only register** that does **not** have **an internal address**.

➤ **DPTR** (data pointer)

- Made up of **two 8-bit registers**, named **DPH** and **DPL**.
- 8051's only user-accessible 16-bit (2-byte) register.
- Provides **addresses** for **internal and external code access** and **external data access**.
- DPH and DPL are **each** assigned an **address**.

16-bit register



Inside the 8051

MOV Instruction

MOV destination, source ; copy source to dest.

- The instruction tells the **CPU** to move (in reality, **COPY**) the **source operand** to the **destination operand**. The MOV instruction **does not affect the source operand**.

“#” signifies that it is a value

MOV A,#55H	;load value 55H into reg. A
MOV R0,A	<u>;copy contents of A into R0</u>
	;(now A=R0=55H)
MOV R1,A	;copy contents of A into R1
	;(now A=R0=R1=55H)
MOV R2,A	;copy contents of A into R2
	;(now A=R0=R1=R2=55H)
MOV R3,#95H	<u>;load value 95H into R3</u>
	;(now R3=95H)
MOV A,R3	;copy contents of R3 into A
	;now A=R3=95H

#55H

Pound sign: #

Compare:

MOV A, #55H

MOV A, 55H

MOV A, #55



Inside the 8051

MOV Instruction

Notes on programming

- Value (preceded with #) can be loaded **directly** to **registers A, B, or R0 – R7**. However, to indicate that it is **an immediate value** it must be preceded with a pound sign (#).
- If values 0 to F moved into an 8-bit register, the rest of the bits are assumed all zeros
 - `MOV A, #5` ; the result will be **A=05**, i.e., **A = 00000101B**
- Moving a value that is too large into a register will cause an error
 - `MOV A, #7F2H` ; **ILLEGAL**: 7F2H > 8 bits (FFH)
 - **Warning: some assemblers don't report this error.**

Common Error

`MOV A, #23H`

`MOV R5, #0F9H`

Add a 0 to indicate that F is a hex number and not a letter

If it's not preceded with #, it means to load from a memory location

`MOV A, 23H`

Valid instruction
Result not be what the programmer intended

`MOV A, #F9H`



Inside the 8051

ADD Instruction

ADD A, source ; ADD the source operand to the Acc.

- ADD instruction tells the CPU to add the source byte to register A and put the result in register A
- Source operand can be either a register or immediate data, but the destination must always be register A
 - “ADD R4, A” and “ADD R2, #12H” are invalid since A must be the destination of any arithmetic operation

25H+34H=?

```
MOV A, #25H    ;load 25H into A
MOV R2, #34H   ;load 34H into R2
ADD A, R2      ;add R2 to Accumulator (A = A + R2)
;----- or -----
MOV A, #25H    ;load one operand into A (A=25H)
ADD A, #34H    ;add the second operand 34H to A
```

There are always many ways to write the same program, depending on the registers used

Introduction to 8051 assembly programming

- In the early days of the computer, programmers coded in **machine language**, consisting of **0s and 1s**
 - Tedious, slow and prone to error
- **Assembly languages**, referred to as a **low-level** language, which provided **mnemonics** for the **machine code** instructions, plus other features
 - Faster and less prone to error
 - Consist of a series of lines of Assembly language instructions
 - Must be translated into machine code (also called *object code*, or *opcode for operation code*) by a program called an **assembler**
 - Deals **directly** with the internal structure of the **CPU**. **Programmer** must know **all the registers of the CPU**, the size of each, and other details.
- C, C++, Java, etc. are called **high-level** languages
 - Does not have to be concerned with the internal details of the CPU
 - Must be translated into machine code by a program called an **compiler**

Introduction to 8051 assembly programming

Structure of Assembly Language

- Assembly language instruction includes
 - **A mnemonic** (abbreviation easy to remember): The commands to the **CPU**, telling it what those to do with those items
 - Optionally followed by **one or two operands: the data** items being manipulated
- A given Assembly language program is a series of statements, or lines
 - Assembly language **instructions指令** such as **MOV** and **ADD**: Tell the **CPU** what to do
 - **Directives** (or **pseudo-instructions, 伪指令**) such as **ORG** and **END**: Give directions to the **assembler**.
 - ✓ **ORG 0H**: tell the assembler to place the opcode at memory location 0
 - ✓ **END**: indicate to the assembler the end of the source code

Introduction to 8051 assembly programming

Structure of Assembly Language

An **Assembly language instruction** consists of **four** fields:

[label:] Mnemonic [operands] [;comment]

Directives do not generate any machine code and are used only by the assembler

Mnemonics produce opcodes

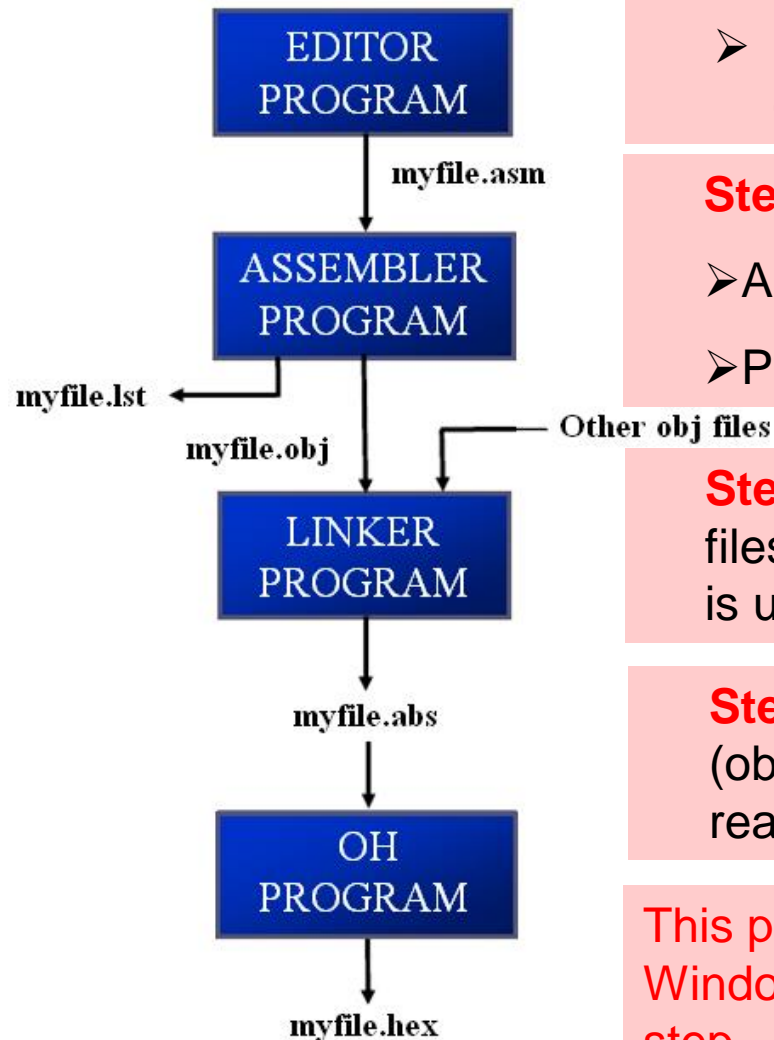
Comments may be at the end of a line or on a line by themselves. The assembler ignores comments

```
ORG 0H           ; start(origin) at location 0
MOV R5, #25H     ; load 25H into R5
MOV R7, #34H
MOV A, #0        ; load 0 into Acc.
ADD A, R5        ; add contents, now A=A+R5
ADD A, R7
ADD A, #12
HERE: SJMP HERE  ; stay in this loop
END              ; a pseudo instruction to tell assembler done.
```

The label field allows the program to refer to a line of code by name; A label must be followed by a colon symbol

ASSEMBLING AND RUNNING AN 8051 PROGRAM

Steps to create a program



Step 1. Use an editor to type a program

- Editor must be able to produce an ASCII file
- Source file has the extension “asm” or “src”, depending on which assembly you are using

Step 2. “asm” source file fed to assembler

- Assembler converts instructions into machine code
- Produce an object file (.obj) and a list file (.lst)

Step 3. Linker program takes one or more object code files and produce an absolute object file (.abs). abs file is used by 8051 trainers that have a monitor program

Step 4. “abs” file is fed into a program called “OH” (object to hex converter) : creates a file (.hex) that is ready to burn into ROM

This program comes with all 8051 assemblers. Recent Windows-based assemblers combine steps 2~4 into 1 step

ASSEMBLING AND RUNNING AN 8051 PROGRAM

Disassembly / CPU Window / Ist file

- The Ist (list) file, which is optional, is very useful to the programmer
 - It lists all the **opcodes** and **addresses** as well as **errors** that the assembler detected
 - The programmer uses the Ist file to find the **syntax errors** or **debug**

MYTEST.ASM		Disassembly	
0000H	7D25	MOV R5, #25H	; MAIN: MOV R5, #25H ; load 25H into R5
0002H	7F34	MOV R7, #34H	; MOV R7, #34H
0004H	7400	MOV A, #00H	; MOV A, #0 ; load 0 into Acc.
0006H	2D	ADD A, R5	; ADD A, R5 ; add contents, now A=A+R5
0007H	2F	ADD A, R7	; ADD A, R7
0008H	240C	ADD A, #0CH	; ADD A, #12
➤ 000AH	80FE	SJMP 000AH	; HERE: SJMP HERE ; stay in this loop

ASSEMBLING AND RUNNING AN 8051 PROGRAM

MOV R5, #data opcode 7D and operand data

1	0000		ORG 0H	;start (origin) at 0
2	0000	7D25	MOV R5, #25H	;load 25H into R5
3	0002	7F34	MOV R7, #34H	;load 34H into R7
4	0004	7400	MOV A, #0	;load 0 into A
5	0006	2D	ADD A, R5	;add contents of R5 to A ;now A = A + R5
6	0007	2F	ADD A, R7	;add contents of R7 to A ;now A = A + R7
7	0008	2412	ADD A, #12H	;add to A value 12H ;now A = A + 12H
8	000A	80EF	HERE: SJMP HERE	;stay in this loop
9	000C		END	;end of asm source file

address

List file

Program counter and ROM space

Program counter

- The program counter points to the **address** of the **next instruction** to be executed
 - As the CPU fetches the opcode from the program ROM, the program counter is increasing to point to the next instruction
- The program counter is **16 bits** wide
 - This means that it can access **program addresses 0000** to **FFFFH**, a total of 64K bytes of code

Power Up

- **All 8051 members start at memory address 0000 when they're powered up**
 - Program Counter has the value of 0000
 - The first opcode is burned into ROM address 0000H, since this is where the 8051 looks for the first instruction when it is booted
 - We achieve this by the **ORG statement** in the source program

Program counter and ROM space

Placing Code in ROM

- Examine the list file and how the code is placed in ROM

ROM Address	Machine Language	Assembly Language
0000	7D25	MOV R5, #25H
0002	7F34	MOV R7, #34H
0004	7400	MOV A, #0
0006	2D	ADD A, R5
0007	2F	ADD A, R7
0008	2412	ADD A, #12H
000A	80EF	HERE: SJMP HERE

Program counter and ROM space

Placing Code in ROM

- **After the program is burned into ROM, the opcode and operand are placed in ROM memory location starting at 0000**

ROM contents

Address	Code
0000	7D
0001	25
0002	7F
0003	34
0004	74
0005	00
0006	2D
0007	2F
0008	24
0009	12
000A	80
000B	FE

CODE	0000	0010	0020	0030	0040	0050	0060	0070	0080	0090	00A0	00B0
	7D 25 7F 34 74 00 2D 2F 24 0C 80 FE FF FF FF FF }%.4t.-/\$.....	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF

地址: 017FH

CODE XDATA

Step 1. When 8051 is **powered up**, the **PC** has **0000H** and starts to fetch the first **opcode** from location **0000H** of program ROM

- Upon executing the opcode 7DH, the CPU fetches the value 25H and places it in R5
- Now one instruction is finished, and then the PC is incremented to point to 0002H, containing opcode 7FH

Step 2. Upon executing the opcode 7FH, the value 34H is moved into R7

- The PC is incremented to 0004H

Step 3. The instruction at location 0004H is executed and now PC = 0006H

Step 4. After the execution of the 1-byte instruction at location 0006H, PC = 0007H

Step 5. Upon execution of this 1-byte instruction at 0007H, PC is incremented to 0008H

- This process goes on until all the instructions are fetched and executed
- The fact that program counter points at the next instruction to be executed explains some microprocessors call it the **instruction pointer**

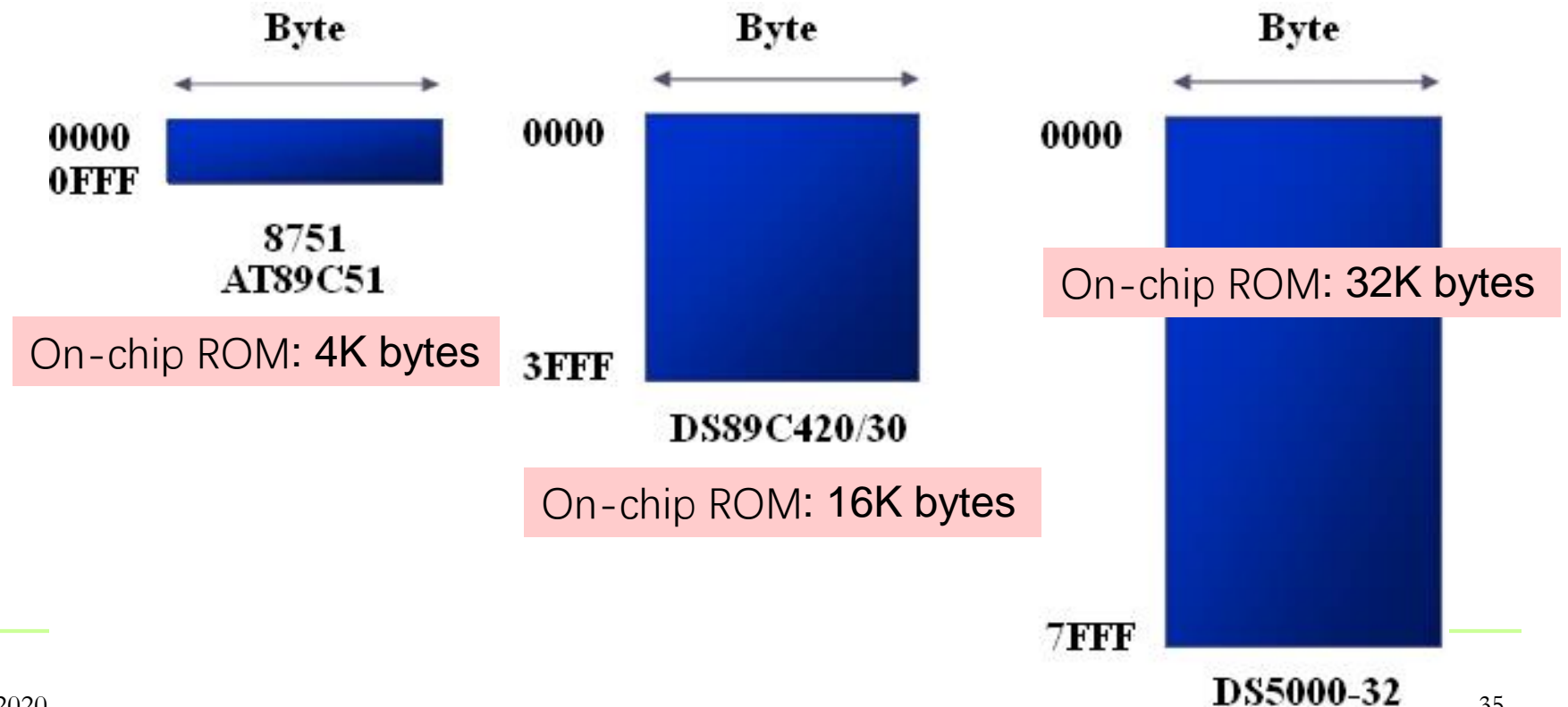
Program counter and ROM space

ROM Memory Map in the 8051 Family

- No member of 8051 family can access more than **64K bytes** of opcode
 - The program counter is a **16-bit** register

The first location of program ROM has the address of **0000H**.

The last location can be different depending on the size of the **ROM on the chip**



8051 data type and directives

Data Type

- 8051 microcontroller has only **one data type** - 8 bits
 - The size of each register is also 8 bits
 - It is the job of the programmer **to break down** data **larger than** 8 bits (00 to FFH, or 0 to 255 in decimal)
 - The data types can be **positive** or **negative**

8051 data type and directives

Assembler Directives

- The **DB** directive is the most widely used data directive in the assembler
 - Used to define the **8-bit data**
 - When DB is used to define data, the numbers can be in **decimal, binary, hex, ASCII formats**

```
ORG 200H  
DATA1:  DB 28      ;DECIMAL (1C in Hex)  
DATA2:  DB 00110101B ;BINARY (35 in Hex)  
DATA3:  DB 39H     ;HEX  
ORG 210H  
DATA4:  DB "2591"  ;ASCII NUMBERS  
ORG 118H  
DATA6:  DB "My name is Joe" ;ASCII CHARACTERS
```

01F0	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0200	1C 35 39 FF FF FF FF FF FF FF FF FF FF FF FF FF	.59.....
0210	32 35 39 31 FF FF FF FF 4D 79 20 6E 61 6D 65 20	2591....My name
0220	69 73 20 4A 6F 65 FF FF FF FF FF FF FF FF FF FF	is Joe.....
0230	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0240	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0250	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0260	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF

地址: 0104H

CODE XDATA

8051 data type and directives

Assembler Directives

ORG (origin)

- The ORG directive is used to indicate **the beginning of the address**
- The number that comes after ORG can **be either in hex and decimal**
 - If the number is not followed by H, it is decimal and the assembler will convert it to hex

END

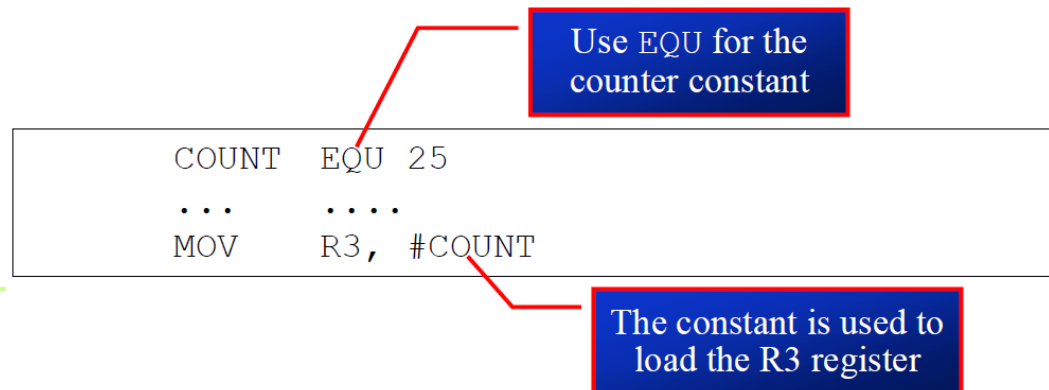
- This indicates to the assembler **the end of the source (asm) file**
- The END directive is **the last line** of an 8051 program
 - Mean that in the code anything after the END directive is ignored by the assembler

8051 data type and directives

Assembler Directives

EQU (equate)

- Used to define a **constant** without occupying a memory location
- The EQU directive does not set aside storage for a data item but associates **a constant value** with **a data label**
 - When the label appears in the program, its constant value will be **substituted** for the label
- Assume that there is a constant used in **many different places** in the program, and the programmer **wants to change its value** throughout
 - By the use of EQU, one can change it **once** and the assembler will change all of its occurrences



8051 data type and directives

Rules for labels in Assembly Language

- Choosing label names are meaningful, much easier to read and maintain
- Several rules that names must follow
 - Each label name must be **unique**
 - The **first character** of the label must be an **alphabetic character**
 - Every assembler has some reserved words that must not be used as labels in the program. Foremost among the reserved words are the mnemonics for the instructions such MOV and ADD

8051 FLAG BITS AND PSW REGISTER

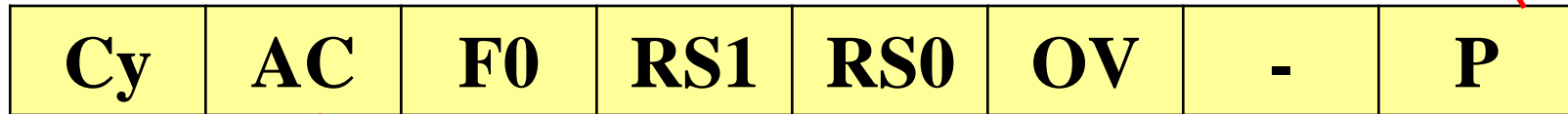
Program Status Word

- The program status word (**PSW**) register, also referred to as the **flag register**, is an **8 bit register**
 - Only **6 bits** are used
 - ✓ **Four of them are** called **conditional flags**, meaning that they indicate some conditions that resulted after an instruction was executed:
 - ❑ **CY** (carry),
 - ❑ **AC** (auxiliary carry),
 - ❑ **P** (parity),
 - ❑ **OV** (overflow)
 - ✓ The **PSW.3** and **PSW.4** are designated as **RS0** and **RS1**, and are used to change the bank registers
 - The two unused bits are user-definable: PSW.5 and PSW.1

8051 FLAG BITS AND PSW REGISTER

Program Status Word

Reflect the number of 1s in register A:
odd number, P=1, otherwise P=0



A carry from D7

A carry from D3 to D4

The result of a signed number operation is too large, causing the high-order bit to overflow into the sign bit

CY	PSW.7	Carry flag.
AC	PSW.6	Auxiliary carry flag.
--	PSW.5	Available to the user for general purpose
RS1	PSW.4	Register Bank selector bit 1.
RS0	PSW.3	Register Bank selector bit 0.
OV	PSW.2	Overflow flag.
--	PSW.1	User definable bit.
P	PSW.0	Parity flag. Set/cleared by hardware each instruction cycle to indicate an odd/even number of 1 bits in the accumulator .

RS1	RS0	Register Bank	Address
0	0	0	00H – 07H
0	1	1	08H – 0FH
1	0	2	10H – 17H
1	1	3	18H – 1FH

Cy flag is used to detect errors in **unsigned** arithmetic operations
OV flag is used to detect errors in **signed** arithmetic operations

8051 FLAG BITS AND PSW REGISTER

ADD Instruction and PSW

Instructions that affect flag bits

Instruction	CY	OV	AC
ADD	X	X	X
ADDC	X	X	X
SUBB	X	X	X
MUL	0	X	
DIV	0	X	
DA	X		
RPC	X		
PLC	X		
SETB C	1		
CLR C	0		
CPL C	X		
ANL C, bit	X		
ANL C, /bit	X		
ORL C, bit	X		
ORL C, /bit	X		
MOV C, bit	X		
CJNE	X		

8051 FLAG BITS AND PSW REGISTER

ADD Instruction and PSW

➤ The flag bits affected by the ADD instruction are **CY**, **P**, **AC**, and **OV**

Example 2-2

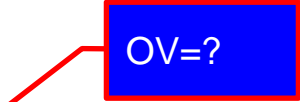
Show the status of the CY, AC and P flag after the addition of 38H and 2FH in the following instructions.

```
MOV A, #38H
```

```
ADD A, #2FH ;after the addition A=67H, CY=0
```

Solution:

38	00111000
+ 2F	<u>00101111</u>
67	01100111



CY = 0 since there is no carry beyond the D7 bit

AC = 1 since there is a carry from the D3 to the D4 bit

P = 1 since the accumulator has an odd number of 1s (it has five 1s)

8051 FLAG BITS AND PSW REGISTER

ADD Instruction and PSW

Example 2-3

Show the status of the CY, AC and P flag after the addition of 9CH and 64H in the following instructions.

```
MOV A, #9CH
```

```
ADD A, #64H ;after the addition A=00H, CY=1
```

Solution:

9C	10011100
+ 64	<u>01100100</u>
100	00000000

OV=?

CY = 1 since there is a carry beyond the D7 bit

AC = 1 since there is a carry from the D3 to the D4 bit

P = 0 since the accumulator has an even number of 1s (it has zero 1s)

8051 FLAG BITS AND PSW REGISTER

ADD Instruction and PSW

Example 2-4

Show the status of the CY, AC and P flag after the addition of 88H and 93H in the following instructions.

```
MOV A, #88H
```

```
ADD A, #93H ;after the addition A=1BH, CY=1
```

Solution:

88	10001000	
+ 93	<u>10010011</u>	
11B	00011011	OV=?

CY = 1 since there is a carry beyond the D7 bit

AC = 0 since there is no carry from the D3 to the D4 bi

P = 0 since the accumulator has an even number of 1s (it has four 1s)

8051 REGISTER BANK AND STACK

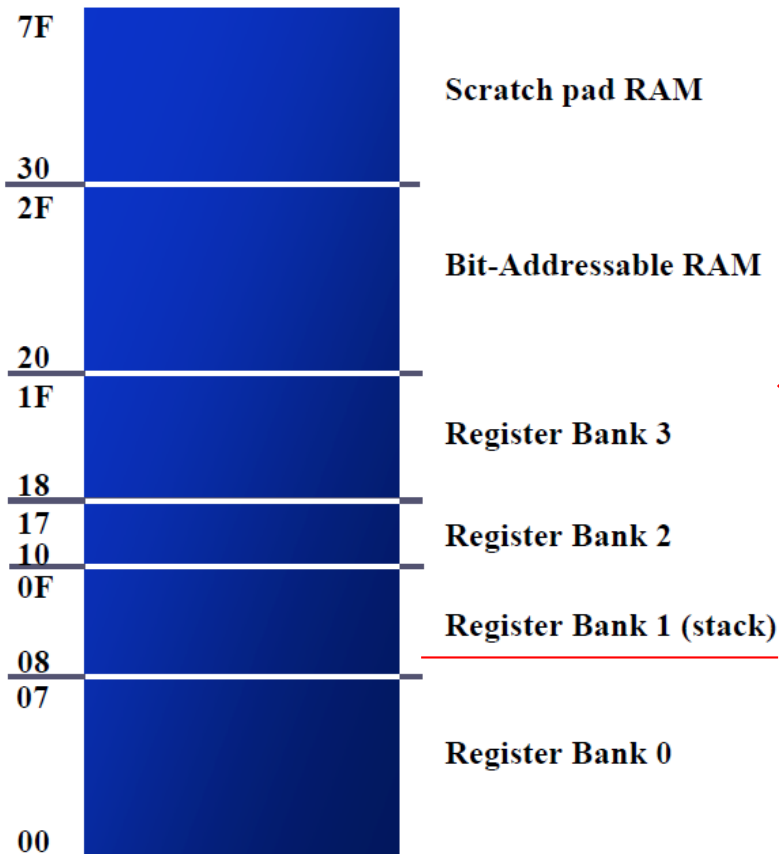
RAM Memory Space Allocation in 8051

- There are **128 bytes of RAM** in the 8051
 - Assigned addresses **00 to 7FH**
- The 128 bytes are divided into **three different groups** as follows:
 - Group 1: A total of 32 bytes from locations **00 to 1F** hex are set aside for **register banks and the stack**.
 - Group 2: A total of 16 bytes from locations **20H to 2FH** are set aside for **bit-addressable read/write memory**.
 - Group 3: A total of 80 bytes from locations **30H to 7FH** are used for read and write storage, called **scratch pad**.

8051 REGISTER BANK AND STACK

RAM Memory Space Allocation in 8051

RAM Allocation in 8051



Register banks and their RAM address

	Bank 0	Bank 1	Bank 2	Bank 3
7	R7	F R7	17 R7	1F R7
6	R6	E R6	16 R6	1E R6
5	R5	D R5	15 R5	1D R5
4	R4	C R4	14 R4	1C R4
3	R3	B R3	13 R3	1B R3
2	R2	A R2	12 R2	1A R2
1	R1	9 R1	11 R1	19 R1
0	R0	8 R0	10 R0	18 R0

8051 REGISTER BANK AND STACK

Register Banks

- These 32 bytes are divided into **4 banks** of registers in which each bank has 8 registers, **R0-R7**
 - RAM location from **0 to 7** are set aside for **bank 0 of R0-R7** where R0 is RAM location 0, R1 is RAM location 1, R2 is RAM location 2, and so on, until memory location 7 which belongs to R7 of bank 0
 - It is much easier to refer to these RAM locations with names such as R0, R1, and so on, than by their memory locations
- Register bank 0 is **the default** when 8051 is powered up
- We can **switch to other banks** by use of the **PSW** register
 - Bits D4 and D3 of the PSW are used to select the desired register bank
 - Use the bit-addressable instructions SETB and CLR to access PSW.4 and PSW.3

PSW bank selection

	RS1(PSW.4)	RS0(PSW.3)
Bank 0	0	0
Bank 1	0	1
Bank 2	1	0
Bank 3	1	1

8051 REGISTER BANK AND STACK

Register Banks

Example 2-5

```
MOV R0, #99H      ;load R0 with 99H
MOV R1, #85H      ;load R1 with 85H
```

Example 2-6

```
MOV 00, #99H      ;RAM location 00H has 99H
MOV 01, #85H      ;RAM location 01H has 85H
```

Example 2-7

```
SETB PSW.4        ;select bank 2
MOV R0, #99H      ;RAM location 10H has 99H
MOV R1, #85H      ;RAM location 11H has 85H
```

8051 REGISTER BANK AND STACK

Stack

- The stack is a section of RAM used by the CPU to store information temporarily
 - This information could be data or an address
- The register used to access the stack is called the SP (stack pointer) register
 - The stack pointer in the 8051 is only 8-bit wide, which means that it can take value of 00 to FFH
 - When the 8051 is powered up, the SP register contains value 07:
 - ✓ RAM location 08 is the first location begin used for the stack by the 8051

8051 REGISTER BANK AND STACK

Stack

- The storing of a CPU register in the stack is called a **PUSH**
 - SP is pointing to the last used location of the stack
 - As we push data onto the stack, the SP is incremented by one
 - ✓ This is different from many microprocessors
- Loading the contents of the stack back into a CPU register is called a **POP**
 - With every pop, the top byte of the stack is copied to the register specified by the instruction, and the stack pointer is decremented once

8051 REGISTER BANK AND STACK

Stack

Example 2-8

Show the stack and stack pointer from the following. Assume the default stack area.

```
MOV R6, #25H
MOV R1, #12H
MOV R4, #0F3H
PUSH 6
PUSH 1
PUSH 4
```

Solution:

	After PUSH 6	After PUSH 1	After PUSH 4																																
<table><tr><td>0B</td><td></td></tr><tr><td>0A</td><td></td></tr><tr><td>09</td><td></td></tr><tr><td>08</td><td></td></tr></table>	0B		0A		09		08		<table><tr><td>0B</td><td></td></tr><tr><td>0A</td><td></td></tr><tr><td>09</td><td></td></tr><tr><td>08</td><td>25</td></tr></table>	0B		0A		09		08	25	<table><tr><td>0B</td><td></td></tr><tr><td>0A</td><td></td></tr><tr><td>09</td><td>12</td></tr><tr><td>08</td><td>25</td></tr></table>	0B		0A		09	12	08	25	<table><tr><td>0B</td><td></td></tr><tr><td>0A</td><td>F3</td></tr><tr><td>09</td><td>12</td></tr><tr><td>08</td><td>25</td></tr></table>	0B		0A	F3	09	12	08	25
0B																																			
0A																																			
09																																			
08																																			
0B																																			
0A																																			
09																																			
08	25																																		
0B																																			
0A																																			
09	12																																		
08	25																																		
0B																																			
0A	F3																																		
09	12																																		
08	25																																		
Start SP = 07	SP = 08	SP = 09	SP = 0A																																

8051 REGISTER BANK AND STACK

Stack

Example 2-9

Examining the stack, show the contents of the register and SP after execution of the following instructions. All value are in hex.

```
POP      3      ; POP stack into R3
POP      5      ; POP stack into R5
POP      2      ; POP stack into R2
```

Solution:

	After POP 3	After POP 5	After POP 2
0B 54	0B	0B	0B
0A F9	0A F9	0A	0A
09 76	09 76	09 76	09
08 6C	08 6C	08 6C	08 6C
Start SP = 0B	SP = 0A	SP = 09	SP = 08

Because locations 20-2FH of RAM are reserved for bit-addressable memory, so we can change the SP to other RAM location by using the instruction "MOV SP, #XX"

24bytes: 08 to 1FH. More than 24 bytes of stack, could change the SP to point 30-7FH by MOV SP, #XX

8051 REGISTER BANK AND STACK

CALL Instruction and the Stack

- The CPU also uses the stack to save the address of the instruction just below the **CALL instruction**
 - This is how the CPU knows where to resume when it returns from the called subroutine

8051 REGISTER BANK AND STACK

Incrementing Stack Pointer

- The reason of **incrementing** SP after push is
 - Make sure that the stack is growing **toward RAM location 7FH**, from lower to upper addresses
 - **Ensure** that the stack will **not reach the bottom of RAM** and consequently run out of stack space
 - If the stack pointer were decremented after push
 - ✓ We would be using RAM locations 7, 6, 5, etc. which belong to R7 to R0 of bank 0, the default register bank.

Stack and Bank 1 Conflict

- When 8051 is powered up, **register bank 1** and the **stack** are using **the same memory space**
 - We can reallocate another section of RAM to the stack

8051 REGISTER BANK AND STACK

Incrementing Stack Pointer

Example 2-10

Examining the stack, show the contents of the register and SP after execution of the following instructions. All value are in hex.

```
MOV SP, #5FH    ;make RAM location 60H
                ;first stack location

MOV R2, #25H
MOV R1, #12H
MOV R4, #0F3H
PUSH 2
PUSH 1
PUSH 4
```

Solution:

	After PUSH 2	After PUSH 1	After PUSH 4																																
<table><tr><td>63</td><td></td></tr><tr><td>62</td><td></td></tr><tr><td>61</td><td></td></tr><tr><td>60</td><td></td></tr></table>	63		62		61		60		<table><tr><td>63</td><td></td></tr><tr><td>62</td><td></td></tr><tr><td>61</td><td></td></tr><tr><td>60</td><td>25</td></tr></table>	63		62		61		60	25	<table><tr><td>63</td><td></td></tr><tr><td>62</td><td></td></tr><tr><td>61</td><td>12</td></tr><tr><td>60</td><td>25</td></tr></table>	63		62		61	12	60	25	<table><tr><td>63</td><td></td></tr><tr><td>62</td><td>F3</td></tr><tr><td>61</td><td>12</td></tr><tr><td>60</td><td>25</td></tr></table>	63		62	F3	61	12	60	25
63																																			
62																																			
61																																			
60																																			
63																																			
62																																			
61																																			
60	25																																		
63																																			
62																																			
61	12																																		
60	25																																		
63																																			
62	F3																																		
61	12																																		
60	25																																		
Start SP = 5F	SP = 60	SP = 61	SP = 62																																

תודה
Dankie Gracias
Спасибо شُكراً
Köszönjük Merci Takk
Grazie Dziękujemy Terima kasih
Ďakujeme Vielen Dank Paldies
Kiitos Tänne teid 谢谢
Thank You Tak
感謝您 Obrigado Teşekkür Ederiz
Σας Ευχαριστούμ 감사합니다
Bedankt Děkuje vám
ありがとうございます
Tack